Seminar

Trends in Model Checking

An Introduction to Robust Markov Decision Processes

Maximizing Rewards under Uncertainty

Peter Lindner
Supervisor: Darion Haase

Summer Semester 2025

1 Introduction

Markov decision processes (MDPs) are a widely used formalism for modeling *probabilistic* and *nondeterministic* systems. They play a central role in research areas such as formal methods [5, 16, 8], artificial intelligence [42], and operations research [28], where they are used for tasks such as decision-making and verification. Standard MDPs assume that transition probabilities are known exactly, which is often an unrealistic assumption [29, 4]. In formal methods, uncertainty may arise from abstraction when models are too large to be analyzed directly [3]. In AI, techniques like reinforcement learning inherently involve uncertainty, as the models they rely on are learned from data [42]. To account for this, *robust* Markov decision processes (RMDPs) have been introduced. These extend standard MDPs by incorporating uncertainty in the transition probabilities, subject to constraints defined by *uncertainty sets*.

In this paper, we build upon the work of Suilen et al. [39] and consider the *reach-reward* objective, where the aim is to maximize the expected cumulative reward until reaching a set of target states. While their paper presents a concise treatment of the subject, our focus is on making these ideas more accessible by systematically presenting the required background and illustrating key concepts through a concrete running example. For this, we present classical dynamic programming techniques for MDPs, and then explore how to adapt them to solve for this objective in the robust setting. Here, the uncertainty is resolved in a worst-case fashion by an adversarial *nature*, and the goal is to synthesize an *optimal scheduler* under such pessimistic assumptions. This yields a lower bound on the maximal expected cumulative reward. We focus on two important instances of RMDPs: *interval MDPs* (IMDPs) and L_1-*MDPs*, whose uncertainty sets satisfy the same structural conditions and are widely studied in both formal methods [17, 22] and AI [27] literature. To illustrate the concepts in practice, we model the running example as both an MDP and an IMDP, and analyze them using the probabilistic model checkers Storm [26] and Prism [31].

The structure of this paper is as follows: Section 2 provides the required preliminaries, including an introduction to MDPs. Section 3 discusses dynamic programming and value iteration for solving the reach-reward objective in MDPs. Section 4 introduces robust MDPs, formalizes their semantics, and presents IMDPs and L_1-MDPs. In Section 5, we show how to adapt the dynamic programming approach to the robust setting and provide algorithms for IMDPs and L_1-MDPs. We conclude in Section 6.

2 Preliminaries

In this section, we introduce the notation used in this paper, followed by the necessary background to understand the contents. We assume familiarity with concepts from probability theory, such as *σ-algebras*, *probability spaces* and *probability measures*. If necessary, consult [30, 5] for a short recap of these concepts. For a set X we denote the cardinality of X as $|X|$. To denote a partial function f with the domain X and codomain Y, we write $f\colon X \dashrightarrow Y$. Let $\mathbb{R}_{\geq 0}$ denote the non-negative reals, and $\mathbb{N}$ the naturals with $0 \in \mathbb{N}$.

Definition 2.1 (Convex Set [12]) *A set $X \subseteq \mathbb{R}^n$ is* convex *if and only if*

$$\forall x_1, x_2 \in X.\ \forall \lambda \in [0,1].\ \lambda x_1 + (1-\lambda)x_2 \in X.$$

Intuitively speaking, a set X is convex if the line segment between any two points $x_1, x_2 \in X$ lies in X. This property typically does not hold if X is discrete.

Given a countable sample space Ω, a function $\mu\colon \Omega \to [0,1]$ such that $\sum_{\omega\in\Omega} \mu(\omega) = 1$ is called a *distribution*. A distribution is referred to as *Dirac* if it assigns probability one to exactly one element and zero to all others. Every distribution μ induces a probability measure $\Pr_\mu\colon \mathcal{F} \to [0,1]$ on the σ-algebra $\mathcal{F} = 2^\Omega$ given by $\Pr_\mu(A) = \sum_{\omega\in A} \mu(\omega)$, for all $A \subseteq \Omega$. We denote the set of all such distributions μ by $\Delta(\Omega)$ [5, 15].

2.1 Probabilistic Models

We now define two kinds of probabilistic models: *Markov decision processes* and a special case of *discrete-time Markov chains*, which form the foundation of the core concepts discussed in this paper. Although the inclusion of the latter may seem surprising, it is required to obtain a probability measure on Markov decision processes that allows us to compute the maximum expected reward until reaching a set of target states. Note that we omit the labeling function from the definitions and instead include the reward function.

Definition 2.2 (Markov Decision Process (MDP) [39]) *A* Markov decision process *is a tuple $\mathcal{M} = (S, s_\iota, A, \mathbf{P}, R)$, where S is a finite set of states, $s_\iota \in S$ is the initial state, A is a finite set of actions, $\mathbf{P}\colon S \times A \dashrightarrow \Delta(S)$ is the probabilistic transition function and $R\colon S \times A \dashrightarrow \mathbb{R}_{\geq 0}$ is the reward function. Moreover, let $A(s) = \{\alpha \in A \mid \boldsymbol{P}(s,\alpha) \neq \bot\}$ be the set of actions available at state s such that $\forall s \in S.\ A(s) \neq \emptyset$.*

Note that in the literature, e.g., [5], Markov decision processes are sometimes equipped with a distribution $\iota_{\text{init}}\colon S \to [0,1]$ over the set of states S instead of a single initial state $s_\iota \in S$, which allows for multiple initial states. However, in this paper we follow the definition of [39] and hence only allow for a single initial state. A transition in an MDP is executed as follows: given a state s, first an action $\alpha \in A(s)$ is chosen *nondeterministically*. Next, a successor state s' is chosen *randomly* according to the probability distribution given by $\mathbf{P}(s,\alpha)$. Thus, after choosing action α one moves from state s to s' with probability $\mathbf{P}(s,\alpha)(s')$ [16]. Together with $A(s) \neq \emptyset$ for every $s \in S$, this means that for every state s there is always some action α available, such that a state $s' \in S$ is reachable with a probability larger than zero. Intuitively speaking, this prevents MDPs from having deadlocks. We require the partial functions $\mathbf{P}$ and R to be defined consistently, meaning $\mathbf{P}(s,\alpha) = \bot \iff R(s,\alpha) = \bot$. Furthermore, we write $\mathbf{P}(s,\alpha,s')$ instead of $\mathbf{P}(s,\alpha)(s')$ for any $s, s' \in S$ and $\alpha \in A$ [39]. Consider Figure 1 as an

example for a Markov decision process, where the labels on the edges encode the action, the corresponding probability and the reward, e.g., for the edge $s_0 \xrightarrow{\alpha} s_1$ we have $\mathbf{P}(s_0, \alpha, s_1) = 1$ and $R(s_0, \alpha) = 0$.

An infinite *path* in an MDP $\mathcal{M}$ is an infinite sequence of states and actions, i.e., $\pi = (s_0, \alpha_0, s_1, \dots) \in (S \times A)^\omega$, such that $\forall n \in \mathbb{N}.\ \mathbf{P}(s_n, \alpha_n, s_{n+1}) > 0$. We denote the set of all infinite paths in $\mathcal{M}$ as $Paths(\mathcal{M})$. A finite prefix $\hat{\pi} = (s_0, \alpha_0, s_1, \alpha_1, \dots, \alpha_{n-1}, s_n) \in (S \times A)^* \times S$ of an infinite path π is called a *finite path fragment* or simply a *finite path*. We write $last(\hat{\pi})$ to describe the last state of a finite path $\hat{\pi}$ and denote the set of all finite paths in $\mathcal{M}$ by $Paths_{fin}(\mathcal{M})$. For any infinite or finite path $\pi = (s_0, \alpha_0, s_1, \alpha_1, \dots)$ of a Markov decision process $\mathcal{M}$, we may write $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \dots$ as a more intuitive notation. Again, consider Figure 1: an example for a finite path would be $\hat{\pi} = s_0\, \alpha\, s_1$, whereas $\pi = s_0\, \alpha\, (s_1\, \beta\, s_2\, \beta\, s_0\, \alpha)^\omega$ is an example for an infinite path in $\mathcal{M}_{\text{ex}}$. A reward $r \in \mathbb{R}_{\geq 0}$ is gained when leaving a state $s \in S$ by taking an action $\alpha \in A$, where $R(s, \alpha) = r$. For example, a reward of $r = 1$ is gained when leaving state s_1 with action β in Figure 1. In the following, we may omit the reward from transitions if it equals zero.

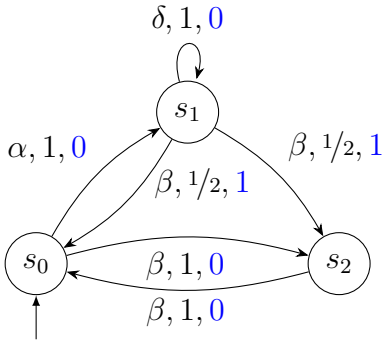


Figure 1: MDP $\mathcal{M}_{\text{ex}}$

Definition 2.3 (Scheduler (Policy, Strategy) [16, 39]) *A* scheduler *for an MDP $\mathcal{M}$ is a function $\sigma\colon Paths_{fin}(\mathcal{M}) \to \Delta(A)$ such that $\sigma(\hat{\pi})(\alpha) > 0$ only if $\alpha \in A(last(\hat{\pi}))$ for $\hat{\pi} \in Paths_{fin}(\mathcal{M})$ and $\alpha \in A$.*

Schedulers defined as above are *history-dependent*, meaning they choose the distribution based on a finite path $\hat{\pi}$. Hence, a history-dependent scheduler σ might choose two different distributions $\mu_1 = \sigma(\hat{\pi}_1), \mu_2 = \sigma(\hat{\pi}_2) \in \Delta(A)$ on two different finite paths $\hat{\pi}_1, \hat{\pi}_2 \in Paths_{fin}(\mathcal{M})$, even if they share the same last state, i.e., $last(\hat{\pi}_1) = last(\hat{\pi}_2)$. We call a scheduler *memoryless* [16, 9] or *stationary* [39] if the resulting distribution $\sigma(\hat{\pi})$ over A depends only on the last state, i.e., $last(\hat{\pi})$, of $\hat{\pi} \in Paths_{fin}(\mathcal{M})$. A scheduler is *deterministic* if it only maps to Dirac distributions. Thus, a memoryless, deterministic scheduler reduces to a function $\sigma\colon S \to A$. Note that depending on the context, schedulers are also known as *policies* [20] or *strategies* [14]. Moreover, schedulers defined as above are sometimes referred to as *randomized* schedulers [5].

Intuitively, a scheduler can be seen as an entity that resolves the nondeterminism of a Markov decision process. This resolution of nondeterminism induces another probabilistic model, called a *discrete-time Markov chain*.

Definition 2.4 (Induced Discrete-Time Markov Chain (DTMC) [39]) *Let $\mathcal{M} = (S, s_\iota, A, \boldsymbol{P}, R)$ be a Markov decision process and $\sigma\colon Paths_{fin}(\mathcal{M}) \to \Delta(A)$ a scheduler. The* induced discrete-time Markov chain *$\mathcal{D}_\sigma$ is a tuple $\mathcal{D}_\sigma = (S^+, s_\iota, \mathbf{P}_\sigma, R_\sigma)$, where S^+ is an (infinite) set of states, s_ι is the initial state, $\mathbf{P}_\sigma\colon S^+ \to \Delta(S^+)$ is the probabilistic transition function and $R_\sigma\colon S^+ \dashrightarrow \mathbb{R}_{\geq 0}$ is the reward function defined for all $\hat{\pi} \in Paths_{fin}(\mathcal{M})$ as*

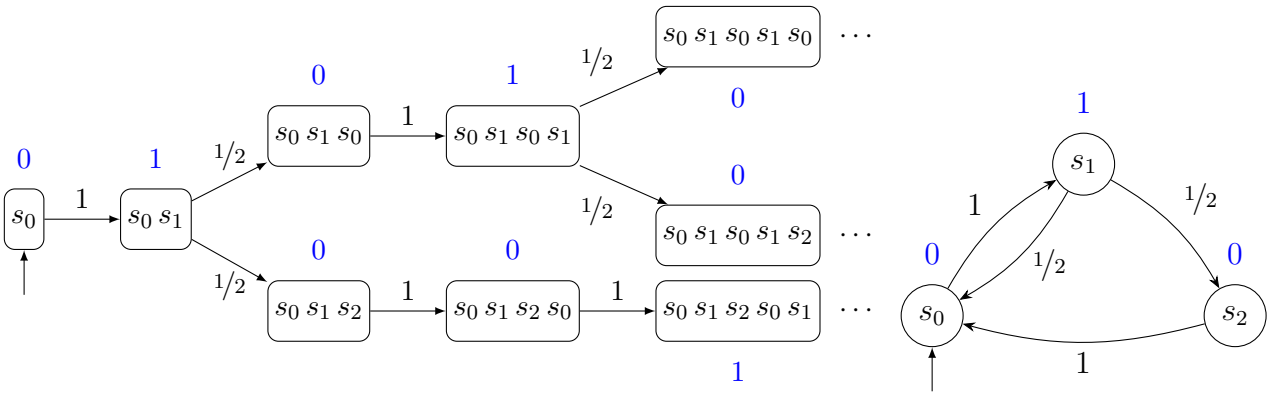
$$\mathbf{P}_\sigma(\hat{\pi}, \hat{\pi}s') = \sum_{\alpha \in A} \sigma(\hat{\pi})(\alpha) \cdot \mathbf{P}(last(\hat{\pi}), \alpha, s')$$

$$R_\sigma(\hat{\pi}) = \sum_{\alpha \in A} \sigma(\hat{\pi})(\alpha) \cdot R(last(\hat{\pi}), \alpha)$$

Initially, it might seem surprising that even though the state space S of the MDP $\mathcal{M}$ is finite, the induced DTMC $\mathcal{D}_\sigma$ can be infinite. Recall that we defined schedulers as history-dependent, hence, they choose distributions depending on the finite path $\hat{\pi}$. On $\mathcal{D}_\sigma$, this history is encoded in the state space, which then turns infinite. For our objective, the focus is on memoryless, deterministic schedulers. Therefore, consider a memoryless, deterministic scheduler σ on the MDP $\mathcal{M}_{\text{ex}}$ given by

$$\sigma(\hat{\pi}) = \begin{cases} \alpha, & \text{if } last(\hat{\pi}) = s_0, \\ \beta, & \text{if } last(\hat{\pi}) = s_1 \vee last(\hat{\pi}) = s_2. \end{cases}$$

The induced DTMC $\mathcal{D}_\sigma$ with an infinite state space S^+ is illustrated in Figure 2a. However, for DTMCs induced by memoryless, deterministic schedulers, the state space can be reduced to a finite one by constructing a *quotient* under *probabilistic bisimulation* [5]. Probabilistic bisimulation is an equivalence relation $\sim_p \subseteq S^+ \times S^+$, which partitions states in equivalence classes based on their behavior. For example, consider the states $(s_0\, s_1), (s_0\, s_1\, s_0\, s_1) \in S^+$ in Figure 2a. Since both have outgoing transitions with the same distribution over the equivalence classes defined by $\sim_p$, and all successors are themselves pairwise related by $\sim_p$, we conclude that $(s_0\, s_1) \sim_p (s_0\, s_1\, s_0\, s_1)$. In our example, this holds for all states that share the same last state. Because every state in an equivalence class behaves identically, we are able to collapse each equivalence class into a single representative state. This yields the quotient under probabilistic bisimulation, denoted by $\mathcal{D}_\sigma/_{\sim_p}$. Hence, for memoryless, deterministic schedulers, we define the induced DTMC $\mathcal{D}_\sigma$ to be given by the quotient $\mathcal{D}_\sigma/_{\sim_p}$. Intuitively, this induced DTMC $\mathcal{D}_\sigma$ can be obtained by omitting all actions from the MDP $\mathcal{M}_{\text{ex}}$ not selected by the scheduler σ; see Figure 2b. For a more thorough explanation, we refer the reader to [5].



(a) DTMC $\mathcal{D}_\sigma$ (b) DTMC $\mathcal{D}_\sigma/_{\sim_p}$

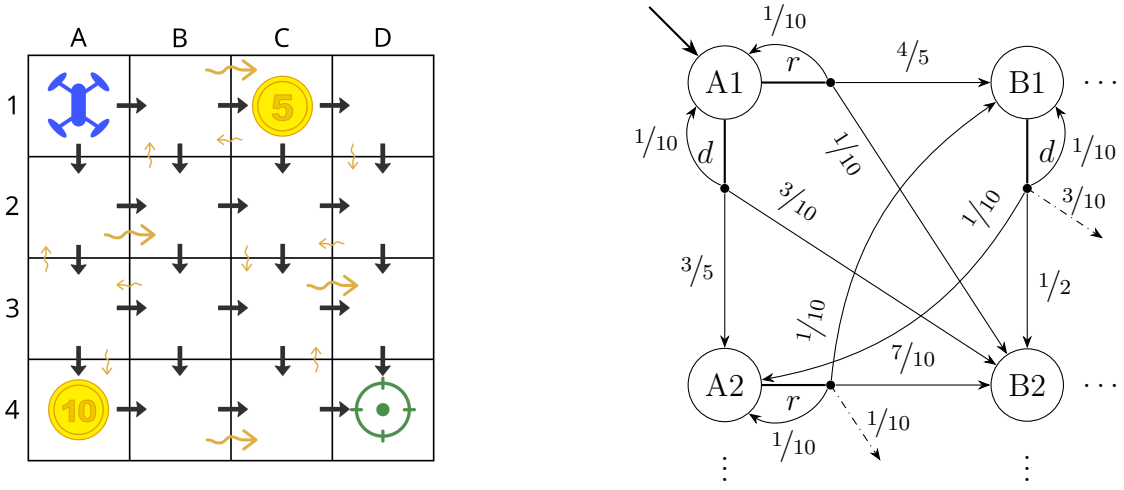
Figure 2: Induced infinite DTMC $\mathcal{D}_\sigma$ and its quotient DTMC $\mathcal{D}_\sigma/_{\sim_p}$ under $\sim_p$ for σ on $\mathcal{M}_{\text{ex}}$

Similarly to MDPs, let $Paths(\mathcal{D}_\sigma)$ denote the set of all infinite paths and $Paths_{fin}(\mathcal{D}_\sigma)$ the set of all finite paths in $\mathcal{D}_\sigma$. Note that the rewards are now collected when leaving a certain state $s \in S$ for which $R(s) \neq \bot$.

Once the nondeterminism is resolved, we can assign probabilities to events in the induced DTMC $\mathcal{D}_\sigma$ by formalizing a probability space over $\mathcal{D}_\sigma$. For more details on how this probability space is constructed, we direct the reader to [5].

3 Solving Reach-Reward in MDPs

In the literature on Markov decision processes, several objectives have been studied, including *reachability* [5, 16], *discounted expected reward* [42], *reach-avoid* [2], general temporal logic objectives expressed by using the probabilistic variants of CTL [19] or LTL [34] and *cumulative reward maximization*. In this paper, as in [39], the focus is on cumulative reward maximization, also referred to as *reach-reward*. In cumulative reward maximization, the objective is to maximize the expected cumulative reward until reaching a set $T \subseteq S$ of target states in an MDP $\mathcal{M}$, as well as determining the corresponding optimal scheduler. We begin with a running example to illustrate the core ideas of this paper in a more intuitive way. Afterwards, we discuss *dynamic programming* and *value iteration* to solve MDPs for our objective.



(a) Schematic overview[1]

(b) Excerpt of MDP $\mathcal{M}_{\text{drone}}$

Figure 3: Drone example

Drone Example. The following example is inspired by the Gymnasium open-source library used to standardize environments in reinforcement learning [43]. Consider the scenario illustrated in Figure 3a, which includes the following main components: a *drone* which moves in a $(n \times n)$-grid, trying to collect as many *deliveries*, which represent rewards, as possible until it reaches the *target* state in the bottom-right corner. Note that the drone is only allowed to move towards the target, i.e., it can only move to the *right* or *down*, while it simultaneously cannot leave the grid. To increase the complexity of the model, it additionally includes *wind*, indicated by the orange arrows in Figure 3a. This wind may cause the drone to drift during movement and eventually places it at an undesired location. The influence of the wind is given by probabilities for each direction. In the following, we denote movements unaffected by the wind as *intended* movements, and call all other movements *affected* or *drift* movements. As an example of a drift movement, assume the drone is located at A1 and wants to move down, i.e., to location A2. If there is wind blowing to the right, this implies a positive probability that the drone ends up in location B2. If the wind blows in the opposite direction of the intended movement, then there is a positive probability for the drone to remain at its current

[1] Created by Richard Lindner, used with permission.

location. Moreover, if the drone is located at the edge of the grid, we ignore the influence of the wind for directions that would force the drone off the grid. For the MDP variant, the wind is assumed to be equal at every position. Furthermore, the probabilities for the wind must be non-zero for each direction. This last restriction ensures graph preservation, meaning that the state space of the model always stays the same, regardless of how strong the wind is. This is required to solve *robust* MDPs, which will be discussed further in Section 5.

Now, the optimal scheduler must choose a *sequence of actions* to maximize the expected cumulative reward until reaching the target state, while accounting for wind effects.

We modeled the example using the PRISM modeling language [1]. The current position of the drone in the grid, whose size can be determined by a constant variable `gridSize`, is encoded by two variables `posX` and `posY`. To ensure that the reward of the deliveries is only collected once, we introduced two Boolean flags `d1`, `d2`. The value and location of the two deliveries, which then give reward on leaving the corresponding location, are encoded in the constants `rewardD1`, `rewardD2` and formulas `atD1`, `atD2`. To model the strength of the wind, we used four variables: `pWindLeft`, `pWindRight`, `pWindUp` and `pWindDown`, whose sum has to be less than one. The model features the two actions `right` and `down`, which change the position of the drone with respect to the drift the wind causes. For more details of the model we refer the reader to our repository [32].

Figure 3b shows a small excerpt of the resulting MDP $\mathcal{M}_{\text{drone}}$ for `gridSize = 4`, `pWindUp` = `pWindLeft` = `pWindDown` = 0.1, `pWindRight` = 0.3, `rewardD1` = 5, `rewardD2` = 10, where the deliveries are located at C1 (delivery one) and A4 (delivery two); see Figure 3a. In the figure, r, d are abbreviations for the actions *right* and *down*.

3.1 Dynamic Programming

Initially, *dynamic programming* was introduced to solve sequential decision-making problems, where the objective is to maximize a function, such as the expected cumulative reward, that depends on the state of the process. This involves evaluating the outcomes of different actions at each decision point based on the current state of the process [9]. In our setting, we consider the problem of computing an *optimal* scheduler for an MDP that maximizes the expected reward accumulated until a set of target states is reached. To accomplish this, we rely on value functions, which intuitively encode how good the current state is with respect to our goal [42].

Preprocessing of the State Space. Before we formulate and solve our objective as a dynamic programming problem, we preprocess the state space S of the MDP $\mathcal{M}$ by analyzing its underlying graph structure. Let $T \subseteq S$ be the set of target states and let $S_?$ be the set of states for which every $s \in S_?$ reaches the T almost surely, i.e., with probability one, under all schedulers. This set $S_?$ can be computed using only backward reachability starting from the set T and some additional bookkeeping. Intuitively, going backwards from the set T, we first exclude all states in S that cannot reach T, and obtain the set $R = S \setminus \{s \in S \mid s \text{ cannot reach } T\}$. Now, we iteratively remove all states from R that have an action leading with positive probability to a state $s \notin R$, and hence obtain our desired set $S_?$ [16]. Trivially, we have $T \subseteq S_?$. While this reduction to the set $S_?$ not only minimizes the amount of work to be done, it also ensures that the value function V^σ yields a finite value for all $s \in S_?$, as we will discuss after formally defining our objective as a dynamic programming problem.

Given a scheduler σ for an MDP $\mathcal{M}$ and a set of target states $T \subseteq S$, the *value function* $V^\sigma \colon S \to \mathbb{R}_{\geq 0}$ assigns to each state $s \in S$ the expected cumulative reward until reaching a state in T. Following our construction that from every state $s \in S_?$ almost surely a state in T is reached, the value function V^σ is the unique least solution [16] to the following recursive equation, known as the *Bellman equation*:

$$V^\sigma(s) = \begin{cases} 0 & \text{if } s \in T, \\ \sum_{\alpha \in A(s)} \sigma(s)(\alpha) \cdot \left[R(s,\alpha) + \sum_{s' \in S_?} \mathbf{P}(s,\alpha,s') \cdot V^\sigma(s') \right] & \text{if } s \in (S_? \setminus T), \\ \infty & \text{if } s \in (S \setminus S_?). \end{cases} \tag{1}$$

The Bellman equation (1) in the case for $s \in (S_? \setminus T)$ describes the relationship between the value of a state and the values of its successor states [42]. Note that for every state $s \in (S \setminus S_?)$, i.e., s does not reach T almost-surely under every possible scheduler, the expected cumulative reward for s either diverges or the computation does not terminate. Thus, we set the value function for all $s \in (S \setminus S_?)$ to ∞.

Now, the goal is to obtain an *optimal* scheduler σ^* such that, for all possible schedulers σ on $\mathcal{M}$, we have:

$$\forall s \in S.\, V^\sigma(s) \leq V^{\sigma^*}(s).$$

Recall, that for the reach-reward objective, memoryless, deterministic schedulers are sufficient [39]. Hence, we are interested in choosing an $\alpha \in A(s)$ for every state $s \in (S_? \setminus T)$ to maximize the expected cumulative reward. To this end, we use *value iteration*, which is a technique to *iteratively* approximate a fixed point. Given a set of states S and a value function $V \colon S \to \mathbb{R}_{\geq 0}$, standard value iteration approximates the fixpoint from below and updates the value function step by step, until a convergence criterion ε is met, i.e., value iteration terminates when $\max_{s \in S} |V^{(n+1)}(s) - V^{(n)}(s)| < \varepsilon$, where $V^{(n)}$ is the value function at iteration n [6, 35, 16]. Note that value iteration does not provide a soundness guarantee, in contrast to *sound value iteration* [36] or *optimistic value iteration* [21]. However, due to its simplicity and as it is still the standard approach for solving MDPs, we utilize it here. To apply value iteration, we first rewrite the Bellman equation (1) in iterative form and replace the sum over all possible actions in s with a max-operator over the set $A(s)$ to consider the optimal action. Hence, we write

$$V^{(n+1)}(s) = \max_{\alpha \in A(s)} \left\{ R(s,\alpha) + \sum_{s' \in S} \mathbf{P}(s,\alpha,s') \cdot V^{(n)}(s') \right\}. \tag{2}$$

With respect to the cases in (1), we now initialize $V(s) = 0$ for all $s \in S_?$ and set $V(s) = \infty$ for all $s \in (S \setminus S_?)$. We then compute the values for all $s \in (S_? \setminus T)$ by applying value iteration and obtain the unique least fixpoint V^* of the Bellman equation (2).

Finally, to obtain the optimal scheduler σ^*, we compute

$$\sigma^*(s) = \arg\max_{\alpha \in A(s)} \left\{ R(s,\alpha) + \sum_{s' \in S} \mathbf{P}(s,\alpha,s') \cdot V^*(s') \right\},$$

which gives the optimal action to be taken in each state $s \in (S_? \setminus T)$ [39].

There are many alternatives to standard value iteration, such as *policy iteration* [16], *sound value iteration* [36], *optimistic value iteration* [21], *linear programming* among others; see [20]

for an overview. Even though value iteration has recently been proven to be EXPTIME-complete by Balaji et al. [7], and hence is theoretically more difficult than other methods, it still tends to perform best on MDPs in practice [20].

To obtain an optimal scheduler of our drone example, we use the probabilistic model checker Storm [26], integrated in the Stormvogel [25] Docker container. We solve the MDP $\mathcal{M}_{\text{drone}}$ for the reach-reward objective by evaluating the property `R{"deliveries"}max=? [ F "reachedTarget" ]`, which yields a list of state-action pairs encoding the optimal scheduler σ^*. In this instance, the optimal expected cumulative reward is approximately 4.29, and the scheduler chooses to move right from the initial state to collect delivery one, despite its lower value. This choice is due to the strong rightward wind (`pWindRight` $= 0.3$), which increases the risk of drifting away when moving toward delivery two. If `pWindRight` is reduced to 0.2, the optimal scheduler instead moves down to collect delivery two, yielding a higher expected reward of approximately 5.25. For more insights, we refer the reader to our repository [32].

4 Robust Markov Decision Processes

After we introduced MDPs and how to solve them with regard to the reach-reward objective, we now focus on MDPs featuring *uncertainty*, so-called *robust* Markov decision processes.

4.1 Theoretical Foundation

Definition 4.1 (Uncertainty Set [39]) *Let X be a set of variables. An* uncertainty set $\mathcal{U}$ *is a non-empty set of functions under constraints, defined as $\mathcal{U} = \{f : X \to \mathbb{R} \mid \text{constraints on } f\}$. These functions represent variable assignments, where each $x \in X$ maps to some $y \in \mathbb{R}$ under specific constraints.*

While the definition of uncertainty sets is intentionally broad, their formulation plays a crucial role in distinguishing different instances of *robust Markov decision processes*, as discussed in Subsection 4.2.

Definition 4.2 (Robust Markov Decision Process (RMDP) [39]) *A* robust Markov decision process *is a tuple $\mathcal{M}_{\mathcal{U}} = (S, s_\iota, A, \boldsymbol{\mathcal{P}}, \mathcal{U}, R)$, where S, s_ι, A, R are defined as for MDPs, $\boldsymbol{\mathcal{P}} : \mathcal{U} \to (S \times A \dashrightarrow \Delta(S))$ is the* uncertain transition function *and $\mathcal{U}$ is an uncertainty set.*

Following the approach in [39], we model uncertainty by introducing a set of variables $X = \{x^\alpha_{s,s'} \mid (s, \alpha, s') \in S \times A \times S\}$, where each variable $x^\alpha_{s,s'}$ corresponds to the transition probability from state s to state s' under action α. Now, the uncertainty set $\mathcal{U}$ constraints the allowed assignments to these variables, where each function $f \in \mathcal{U}$ assigns a value in $[0, 1]$ to every variable $x^\alpha_{s,s'} \in X$. Thus, every $f \in \mathcal{U}$ represents a unique assignment to these variables that satisfies the constraints of $\mathcal{U}$. These constraints may, for example, encode upper and lower bounds or dependencies between transitions. We interpret each assignment $f \in \mathcal{U}$ as inducing a transition function $\boldsymbol{\mathcal{P}}(f)$, defined by $\boldsymbol{\mathcal{P}}(f)(s, \alpha, s') = f(x^\alpha_{s,s'})$. This construction yields a valid probabilistic transition function if and only if f satisfies the necessary constraints in $\mathcal{U}$, such as ensuring $\sum_{s' \in S} f(x^\alpha_{s,s'}) = 1$ for every state-action pair (s, α). Moreover, this allows us to express the constraints defining $f \in \mathcal{U}$ directly over the variables $x^\alpha_{s,s'} \in X$, rather than in terms of their image $f(x^\alpha_{s,s'})$.

We revisit the drone example from Section 3. In Figure 4, we replaced the probabilities of the possible transitions by variables in the set X. Additionally, we colored the transitions representing the intended movements green, and the transitions for the drift movements orange. Note that this excerpt is a simplification of the real model, since the delivery flags are not encoded in the state space. These flags ensure that the rewards earned by, for example, taking the transition marked with $x^r_{\text{A4,A4}}$, are only collected once. When modeling this example as an RMDP, it becomes much more expressive: instead of fixed probabilities for each wind direction, we can now consider intervals, for example. To illustrate this, consider the following uncertainty sets:

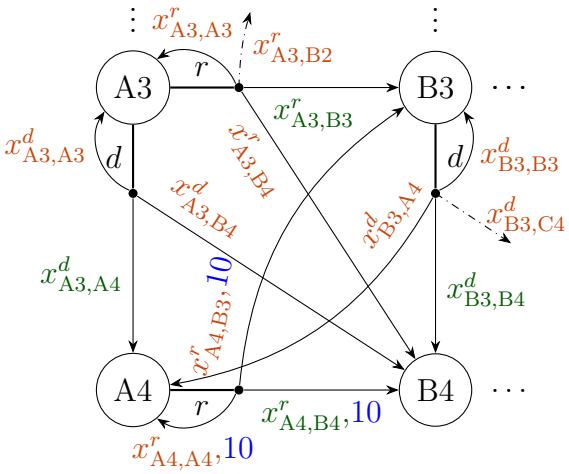


Figure 4: Excerpt of RMDP $\mathcal{M}_{\mathcal{U},\text{drone}}$

$$\mathcal{U}_1 = \{\ f \mid x^r_{\text{A3,B3}}, x^d_{\text{A3,A4}}, x^r_{\text{A4,B4}}, x^d_{\text{B3,B4}}, \ldots \in [0.5, 0.8] \ \wedge\ \varphi_{\text{drift}}\ \} \tag{3}$$

$$\mathcal{U}_2 = \{\ f \mid (x^r_{\mathbf{A3},\text{B3}} = x^d_{\mathbf{A3},\text{A4}} \wedge \ldots) \wedge (x^r_{\text{A3,B3}}, x^d_{\text{A3,A4}}, \ldots \in [0.5, 0.8]) \ \wedge\ \varphi_{\text{drift}}\ \} \tag{4}$$

$$\mathcal{U}_3 = \{\ f \mid (x^r_{\mathbf{A3},\text{B3}} = x^r_{\mathbf{A4},\text{B4}} = x^d_{\text{A3,A4}} = x^d_{\text{B3,B4}} = \ldots) \wedge (x^r_{\text{A3,B3}} \in [0.5, 0.8]) \ \wedge\ \varphi_{\text{drift}}\ \} \tag{5}$$

$$\varphi_{\text{drift}} = x^r_{\text{A3,A3}}, x^r_{\text{A3,B2}}, x^r_{\text{A3,B4}}, x^d_{\text{A3,A3}}, x^d_{\text{A3,B4}}, x^r_{\text{A4,A4}}, x^r_{\text{A4,B3}}, x^d_{\text{B3,B3}}, x^d_{\text{B3,A4}}, x^d_{\text{B3,C4}}, \ldots \in [0.1, 0.3]$$

In all three uncertainty sets, we use φ_{drift} to restrict the variables of the drift movements to lie in the interval $[0.1, 0.3]$. Hence, they only differ in terms of constraints on intended movements. The first uncertainty set $\mathcal{U}_1$ allows all variables for the intended movements to be in the interval $[0.5, 0.8]$. In uncertainty set $\mathcal{U}_2$, we require the variables describing the unaffected left and down movements to be equal for each individual state and to lie within the interval $[0.5, 0.8]$. Lastly, in uncertainty set $\mathcal{U}_3$, all variables of the intended movements to the right or downwards lie in the interval $[0.5, 0.8]$ and must be equal, regardless of the state. Consider the following example functions, each belonging to one of the uncertainty sets:

$$f_1 = \{x^r_{\text{A3,B3}} \mapsto 0.6, x^d_{\text{A3,A4}} \mapsto 0.8, \ldots, x^r_{\text{A3,A3}} \mapsto 0.15, x^d_{\text{A3,B2}} \mapsto 1/6, \ldots\} \qquad \in \mathcal{U}_1$$

$$f_2 = \{x^r_{\mathbf{A3},\text{B3}} = x^d_{\mathbf{A3},\text{A4}} \mapsto 0.6, x^r_{\mathbf{A4},\text{B4}} \mapsto 0.7, \ldots, x^r_{\text{A3,A3}} \mapsto 1/9, x^d_{\text{A3,B2}} \mapsto 0.1, \ldots\} \qquad \in \mathcal{U}_2$$

$$f_3 = \{x^r_{\mathbf{A3},\text{B3}} = x^r_{\mathbf{A4},\text{B4}} \mapsto 0.7, \ldots, x^r_{\text{A3,A3}} \mapsto 0.2, x^d_{\text{A3,B2}} \mapsto 0.13, \ldots\} \qquad \in \mathcal{U}_3$$

In the RMDP variant of the example, we relaxed the restriction for the probabilities of the wind to be constant throughout the grid. Even though we could enforce this restriction through additional constraints, we want the model to be more expressive here.

By defining uncertainty sets this way, we may encode dependencies between variables of the set X. Depending on the structure of these dependencies, we can categorize uncertainty sets based on their *rectangularity*.

Definition 4.3 (Rectangularity [39]) *Let $\mathcal{U}$ be an uncertainty set for an RMDP $\mathcal{M}_{\mathcal{U}}$. We say $\mathcal{U}$ is either*

1. (s,α)-rectangular *iff* $\mathcal{U} = \prod_{(s,\alpha)\in S\times A} \mathcal{U}_{(s,\alpha)}$, *where each* $\mathcal{U}_{(s,\alpha)}$ *is a* lower-dimensional *(*local*) uncertainty set only restricting over the variables* $X_{s,\alpha} = \{x^\alpha_{s,s'} \mid s' \in S\}$;

2. (s)-rectangular *iff* $\mathcal{U} = \bigsqcap_{s \in S} \mathcal{U}_{(s)}$, *where each* $\mathcal{U}_{(s)}$ *is a* lower-dimensional *or* local *uncertainty set only restricting over the variables* $X_s = \bigcup_{\alpha \in A} X_{s,\alpha}$ *or*

3. non-rectangular *otherwise.*

Hence, the classification of an uncertainty set depends on whether it can be split into lower-dimensional uncertainty sets. We use $\bigsqcap$ to denote the intersection of constraints. That is, an (s, α)-rectangular uncertainty set $\mathcal{U}$ contains exactly those functions that satisfy all local constraints stated by each $\mathcal{U}_{(s,\alpha)}$. An analogous interpretation holds for (s)-rectangular uncertainty sets.

Consider the uncertainty sets defined for our drone example above. As $\mathcal{U}_1$ does not encode any dependencies between the variables, but only defines intervals for each, it is (s, α)-*rectangular*. It can be easily split into lower dimensional uncertainty sets for each (s, α)-pair. For $\mathcal{U}_2$, we give dependencies concerning the variables of each state. These dependencies can only be split at state-level, as the dependencies affect different actions. Thus, it represents an (s)-*rectangular* uncertainty set. Lastly, for $\mathcal{U}_3$, we have dependencies ranging between different actions and states, classifying it as a *non-rectangular* uncertainty set.

RMDP Semantics [39]. The semantics of RMDPs can be interpreted as a game between an *agent*, which controls the scheduler by choosing the actions, and an adversarial *nature*. Adversarial in this context means, that nature is trying to minimize the reward based on the choices of the agent by choosing the worst-case variable assignment for each state-action pair by selecting a realization $f \in \mathcal{U}$ under the constraints of the uncertainty set $\mathcal{U}$. In this way, nature constructs the probabilistic transition function $\boldsymbol{\mathcal{P}}(f) = \mathbf{P}$. Nature's strategy in choosing this assignment depends on two properties: *last-action observability* and *static or dynamic uncertainty semantics*.

Last-action observability follows directly from the class of the uncertainty set $\mathcal{U}$. If $\mathcal{U}$ is (s, α)-rectangular, then nature may base its decision on the most-recent action chosen by the agent. This scenario is also referred to as *agent-first* semantics [10]. For (s)-rectangular and non-rectangular uncertainty sets, nature is only able to observe the last state of the agent, which is also referred to as *nature-first* semantics [10].

Static or dynamic uncertainty semantics describe whether nature is allowed to make a different decision than before if it encounters a state-action pair again. Assuming static uncertainty semantics means that once nature has made a decision on a specific state-action pair, it must stick to this variable assignment in the future. This is similar to the concept of a memoryless scheduler, as discussed in Subsection 2.1. Conversely, under dynamic uncertainty semantics, nature may choose a different variable assignment if it encounters the same state-action pair again. In [28], Iyengar shows that for (s, α)-rectangular RMDPs, the distinction between static or dynamic uncertainty semantics has no effect on reward-maximization.

To illustrate this game concept and how a transition is executed in RMDPs, consider the excerpt given in Figure 4 and the uncertainty set $\mathcal{U}_1$ as stated in (3). Recall that $\mathcal{U}_1$ is (s, α)-rectangular, hence the game follows the *agent-first* semantics. Furthermore, we assume that the drone, controlled by the agent, is currently at location A3. To maximize their[2] reward, the agent chooses action d to reach delivery two at location A4. As nature acts adversarial, i.e., trying to minimize the agent's reward, it chooses $f_1 = \{x^d_{\text{A3,A4}} \mapsto 0.5, x^d_{\text{A3,B4}} \mapsto 0.3, x^d_{\text{A3,A3}} \mapsto$

[2]We use the gender-neutral personal pronoun form "they"/"their" in both singular and plural.

$0.2, \ldots\} \in \mathcal{U}_1$, which minimizes the probability for this transition. Note how nature assigns a higher value to the drift transition $x^d_{\text{A3,B4}}$ than to $x^d_{\text{A3,A3}}$, as the model does not allow the drone to move to the left. Thus, after ending up at location B4, the agent misses delivery two completely.

Alternatively, if we consider the uncertainty set $\mathcal{U}_2$ (4) in the scenario given above, the game follows the *nature-first* semantics. Therefore, nature has to choose a variable assignment which minimizes the reward of the agent regardless of their choice of action; essentially limiting its ability to react in an "optimal" way. Thus, nature chooses an assignment $f_2 = \{x^r_{\text{A3,B3}} = x^d_{\text{A3,A4}} \mapsto 0.5, x^d_{\text{A3,B4}} \mapsto 0.3, x^r_{\text{A3,B4}} \mapsto 0.3, \ldots\} \in \mathcal{U}_2$. Since $x^d_{\text{A3,A4}}$ leads directly to the state containing delivery two, nature chooses the smallest probability for the respective transition. Note that following the constraints of $\mathcal{U}_2$, the variables $x^r_{\text{A3,B3}}, x^d_{\text{A3,A4}}$ must be equal. As the transitions encoded by the variables $x^d_{\text{A3,B4}}$ and $x^r_{\text{A3,B4}}$ make it impossible to reach delivery two afterwards, they are assigned the upper bound of their interval.

Once both the nondeterminism (via the agent) and the uncertainty (via nature) have been resolved through this game, the system transitions from state s to s' with probability $\mathbf{P}(s, \alpha, s')$, where $\mathbf{P} = \boldsymbol{\mathcal{P}}(f)$ for some $f \in \mathcal{U}$.

4.2 Well-known Instances

Having introduced the foundations of robust Markov decision processes, we now present two well-established instances frequently used in AI and formal methods: *interval MDPs* (IMDPs) and *L_1-MDPs*. Following Suilen et al. [39], we first give the standard definitions and show how both instances fit within the RMDP framework, including the shape of their respective uncertainty set. While Suilen et al. [39] additionally discuss *multi-environment MDPs* (MEMDPs), an (s)-rectangular RMDP variant representing a family of MDPs with separate transition functions, we omit them here, as our focus is restricted to solving (s, α)-rectangular RMDPs.

4.2.1 Interval-MDPs

Definition 4.4 (Interval Markov Decision Process (IMDP) [22, 17]) *Let $\mathbb{I}$ be a set including all closed subintervals of $[0, 1]$ with strictly positive lower bound, given by $\mathbb{I} = \{[l, u] \mid 0 < l \leq u \leq 1\}$. An* interval Markov decision process *is a tuple $\mathcal{M}^{\mathbb{I}} = (S, s_\iota, A, \mathbf{P}, R)$, where S, s_ι, A, R are defined as for MDPs and $\mathbf{P} \colon S \times A \times S \dashrightarrow \mathbb{I} \cup \{[0, 0]\}$ is the* probabilistic interval transition function.

For IMDPs, every possible transition is assigned an interval instead of a probability. By requiring $l > 0$ in the definition of $\mathbb{I}$, we ensure the preservation of the underlying graph for every possible resolution of the intervals. IMDPs have been thoroughly discussed in the literature, ranging from bisimulation [22, 18, 24, 23] to research on multi-objective synthesis [17]. They are supported by modern tools like Prism [31] and Storm [26], where the former already treats IMDPs as first class citizen. IMDPs are also expressible in the Prism modeling language [1]. To illustrate how IMDPs fit in the RMDP framework, we give a formal definition.

Definition 4.5 (IMDP as RMDP [39]) *An IMDP is an RMDP $\mathcal{M}^{\mathbb{I}}_{\mathcal{U}}$ given by the tuple $\mathcal{M}^{\mathbb{I}}_{\mathcal{U}} = (S, s_\iota, A, \boldsymbol{\mathcal{P}}, \mathcal{U}, R)$, where $\mathcal{U} = \{f \colon X \to \mathbb{R} \mid \forall (s, \alpha, s') \in S \times A \times S.\ x^\alpha_{s,s'} \in [l, u]^\alpha_{s,s'} \subseteq [0, 1] \wedge \forall (s, \alpha) \in S \times A.\ \sum_{s' \in S} x^\alpha_{s,s'} = 1\}$ and $S, s_\iota, A, \boldsymbol{\mathcal{P}}, R$ are defined as for general RMDPs.*

Here, $[l, u]^\alpha_{s,s'} \in \mathbb{I}$ denotes the interval of allowed transition probabilities from state s to state s' under action α. As a result of this definition, IMDPs are (s, α)-rectangular.

Note that the uncertainty set $\mathcal{U}_1$, given in Equation 3, precisely describes our drone example as an IMDP by using the RMDP definition. When exchanging the variables of Figure 4 with their respective intervals, we obtain the graph representation of an IMDP.

4.2.2 L_1-MDPs

Definition 4.6 (L_1-Markov Decision Process (L_1-MDP) [39]) *A* L_1-Markov decision process *is a tuple* $\mathcal{M}^L = (S, s_\iota, A, \tilde{\mathbf{P}}, R, d)$*, where* S, s_ι, A, R *are defined as for MDPs,* $\tilde{\mathbf{P}} \colon S \times A \dashrightarrow \Delta(S)$ *is the* centre transition function *and* $d \colon S \times A \to \mathbb{R}_{\geq 0}$ *is a* distance function *assigning an error to each state-action pair.*

For L_1-MDPs, the centre transition function acts as a reference distribution for which the distance function d bounds the L_1-error. Consider the following definition as an (s, α)-rectangular RMDP.

Definition 4.7 (L_1-MDP as RMDP [39]) *An* L_1*-MDP is an RMDP* $\mathcal{M}^L_{\mathcal{U}}$ *given by the tuple* $\mathcal{M}^L_{\mathcal{U}} = (S, s_\iota, A, \boldsymbol{\mathcal{P}}, \mathcal{U}, R)$*, where* $\mathcal{U} = \{f \colon X \to \mathbb{R} \mid \forall (s, \alpha) \in S \times A. \ \sum_{s' \in S} | x^\alpha_{s,s'} - \tilde{\mathbf{P}}(s, \alpha, s') | \leq d(s, \alpha)\}$ *and* $S, s_\iota, A, \boldsymbol{\mathcal{P}}, R$ *are defined as for general RMDPs.*

5 Solving Reach-Reward in Robust MDPs

To solve the reach-reward objective for (s, α)-rectangular RMDPs, we distinguish between worst-case and best-case resolutions of uncertainty, as there is no single transition function. While Suilen et al. consider both cases in [39], we focus on the worst-case interpretation, meaning that we seek an optimal scheduler that maximizes the objective under the worst possible conditions. This approach yields a lower bound on the expected reward. Conversely, for best-case resolution, also known as *optimistic* resolution, we obtain the upper bound. In [44], Wiesemann et al. identified the optimal scheduler classes with respect to properties of the uncertainty set; see Table 1.

Uncertainty set	**Rectangularity**	**Optimal scheduler class**
Convex	(s, α)-rectangular	Memoryless, deterministic
Convex	(s)-rectangular	Memoryless, randomized
Convex	non-rectangular	History-dependent, randomized
Nonconvex	(s, α)-rectangular	Memoryless, deterministic
Nonconvex	(s)-rectangular	History-dependent, randomized
Nonconvex	non-rectangular	History-dependent, randomized

Table 1: Optimal scheduler classes depending on the uncertainty set according to [44].

5.1 Robust Dynamic Programming

In the following section, we discuss how to solve (s, α)-rectangular RMDPs for reach-reward using *robust* dynamic programming, based on [39]. Recall the reach-reward objective and its formulation as a dynamic programming problem from Subsection 3.1. *Robust* dynamic programming extends the techniques on MDPs to robust MDPs. Hence, we are now given an

(s,α)-rectangular RMDP $\mathcal{M}_\mathcal{U}$, a scheduler σ on $\mathcal{M}_\mathcal{U}$ and a set of target states $T \subseteq S$. Per definition, $\mathcal{M}_\mathcal{U}$ has multiple possible probabilistic transition functions $\boldsymbol{\mathcal{P}}(f) = \mathbf{P}$ depending on a variable assignment $f \in \mathcal{U}$. To ensure computational tractability, we require the uncertainty set $\mathcal{U}$ to be *graph preserving*, i.e., if for some $\boldsymbol{\mathcal{P}}(f) = \mathbf{P}, f \in \mathcal{U}$ there exists a transition such that $\mathbf{P}(s,\alpha,s') = 0$, then for all other $\boldsymbol{\mathcal{P}}(f') = \mathbf{P}', f' \in \mathcal{U}$, $\mathbf{P}'(s,\alpha,s') = 0$ must hold as well. Moreover, we apply the same preprocessing steps to the state space as mentioned in Subsection 3.1 to obtain the set $S_?$. In the following, we slightly abuse notation by writing $\mathbf{P} \in \boldsymbol{\mathcal{P}}$ as a shorthand for obtaining $\boldsymbol{\mathcal{P}}(f) = \mathbf{P}$ for some $f \in \mathcal{U}$. To handle the uncertainty over multiple possible transition functions, we rewrite the Bellman equation (1) with respect to the worst-case interpretation, and obtain

$$\underline{V}^\sigma(s) = \begin{cases} 0 & \text{if } s \in T, \\ \sum\limits_{\alpha \in A(s)} \sigma(s)(\alpha) \cdot \left[R(s,\alpha) + \inf\limits_{\mathbf{P} \in \boldsymbol{\mathcal{P}}} \left\{ \sum_{s' \in S_?} \mathbf{P}(s,\alpha,s') \cdot \underline{V}^\sigma(s') \right\} \right] & \text{if } s \in (S_? \setminus T), \\ \infty & \text{if } s \in (S \setminus S_?). \end{cases} \tag{6}$$

Here, the value function $\underline{V}^\sigma \colon S \to \mathbb{R}_{\geq 0}$ denotes the unique solution [28]. In equation (6), the worst-case interpretation is encoded in taking the infimum over all transition functions.

For RMDPs, we adapt the changes from Subsection 3.1 to *robust* value iteration by introducing the *inner minimization problem* to be solved at each iteration, similarly to (6):

$$\underline{V}^{(n+1)}(s) = \max_{\alpha \in A(s)} \left\{ R(s,\alpha) + \inf_{\mathbf{P} \in \boldsymbol{\mathcal{P}}} \left\{ \sum_{s' \in S} \mathbf{P}(s,\alpha,s') \cdot \underline{V}^{(n)}(s') \right\} \right\}. \tag{7}$$

The complexity of solving this inner minimization problem determines whether an RMDP can be solved efficiently, especially whether the uncertainty sets are convex, as convex optimizations may be applied [39].

For (s,α)-rectangular RMDPs, the uncertainty set $\mathcal{U}$ does not allow for dependencies between different state-actions pairs, as it can be decomposed into lower-dimensional uncertainty sets $\mathcal{U}_{(s,\alpha)}$; see Subsection 4.1. Hence, we can relax the Bellman equation (7), replacing the global minimization problem with a local one that minimizes only over the uncertainty set corresponding to a fixed state-action pair. Thus, for (s,α)-rectangular RMDPs, we may write

$$\underline{V}^{(n+1)}(s) = \max_{\alpha \in A(s)} \left\{ R(s,\alpha) + \inf_{\mathbf{P}(s,\alpha) \in \boldsymbol{\mathcal{P}}(s,\alpha)} \left\{ \sum_{s' \in S} \mathbf{P}(s,\alpha,s') \cdot \underline{V}^{(n)}(s') \right\} \right\}, \tag{8}$$

where $\mathbf{P}(s,\alpha) \in \boldsymbol{\mathcal{P}}(s,\alpha)$ denotes the restriction of the transition function to a local uncertainty set $\mathcal{U}_{(s,\alpha)}$. Note that this form is only valid under the assumption of (s,α)-rectangularity. Applying this simplification to uncertainty sets with a different structure may yield solutions that violate the constraints of the full uncertainty set $\mathcal{U}$.

After computing the fixpoint $\underline{V}^*$ of (8), we obtain the optimal robust scheduler $\underline{\sigma}^*$, which is memoryless and deterministic (see Table 1), by computing

$$\underline{\sigma}^*(s) = \arg\max_{\alpha \in A(s)} \left\{ R(s,\alpha) + \inf_{\mathbf{P}(s,\alpha) \in \boldsymbol{\mathcal{P}}(s,\alpha)} \left\{ \sum_{s' \in S} \mathbf{P}(s,\alpha,s') \cdot \underline{V}^*(s') \right\} \right\}.$$

Although robust value iteration is the standard approach for computing optimal RMDP schedulers, a case study performed by Ho et al. [27] suggests that *robust policy iteration* is more

efficient than robust value iteration, at least for L_1-RMDPs. The authors argue that robust policy iteration requires fewer computations of the inner minimization problem, which may become quite expensive. However, there remains the need for an extensive experimental evaluation that supports this hypothesis [39].

For RMDPs, we modeled our example as an IMDP using the PRISM modeling language. The model extends the MDP variant by introducing lower and upper bounds for each wind probability. All bounds match the fixed values of the original MDP, except for `pWindRight`, which varies within $[0.1, 0.3]$. The bounds for the intended movements are then derived by considering the lower and upper bounds of each wind direction, ensuring the total probability mass remains valid. Together, these form the uncertainty set $\mathcal{U}_{\text{drone}}$. Since STORM does not yet support reach-reward properties for IMDPs, we used PRISM instead. To evaluate the worst-case resolution of uncertainty, we checked the property `R{"deliveries"}maxmin=? [ F "reachedTarget" ]`. Since nature can choose transition probabilities adversarially within the allowed intervals, the agent adopts a more conservative strategy compared to the MDP variant. As a result, the optimal scheduler σ^* moves right toward delivery one in the initial location, yielding a lower-bound expected cumulative reward of approximately 4.29. Although delivery one gives less reward, it is less exposed to the strong rightward drift, making it a safer choice under uncertainty. Note that this coincides with the MDP variant with `pWindRight` $=$ 0.3, as nature may choose the upper bound of the interval at every step if needed. Further details and source code are available in our repository [32].

5.2 Solving the Inner Minimization Problem

To complete our approach of computing an optimal scheduler for (s, α)-rectangular RMDPs, we now focus on solving the inner minimization problem for interval MDPs and L_1-MDPs.

5.2.1 Interval MDPs

To solve the inner minimization problem for IMDPs, consider the bisection algorithm stated in Algorithm 1. This algorithm was originally introduced by Nilim and El Ghaoui [33] and later reformulated by Suilen et al [39]. The goal is to compute $\inf_{\mathbf{P}(s,\alpha)\in\boldsymbol{\mathcal{P}}(s,\alpha)}$ of (8) for a fixed state-action pair (s, α) at iteration n.

Algorithm 1: Algorithm to solve the IMDP inner problem $\inf_{\mathbf{P}(s,\alpha)\in\boldsymbol{\mathcal{P}}(s,\alpha)}$ [39]

Input: Pair (s, α), bounds $\inf \mathbf{P}(s, \alpha, \cdot)$ and $\sup \mathbf{P}(s, \alpha, \cdot)$, value function $\underline{V}^{(n)}$
Sort $S' = \{s'_1, \ldots, s'_m\}$ in ascending order such that $\underline{V}^{(n)}(s'_i) \leq \underline{V}^{(n)}(s'_{i+1})$;
foreach $s'_i \in S'$ **do**
    $\mathbf{P}_{\inf}(s, \alpha, s'_i) \leftarrow \inf \mathbf{P}(s, \alpha, s'_i)$;
$budget \leftarrow 1 - \sum_{s' \in S'} \inf \mathbf{P}(s, \alpha, s')$;
$i \leftarrow 1$;
while $budget - (\sup \mathbf{P}(s, \alpha, s'_i) - \inf \mathbf{P}(s, \alpha, s'_i)) \geq 0$ **do**
    $\mathbf{P}_{\inf}(s, \alpha, s'_i) \leftarrow \sup \mathbf{P}(s, \alpha, s'_i)$;
    $budget \leftarrow budget - (\sup \mathbf{P}(s, \alpha, s'_i) - \inf \mathbf{P}(s, \alpha, s'_i))$;
    $i \leftarrow i + 1$;
$\mathbf{P}_{\inf}(s, \alpha, s'_i) \leftarrow \inf \mathbf{P}(s, \alpha, s'_i) + budget$;
return $\mathbf{P}_{\inf}(s, \alpha, \cdot)$;

Intuitively, the algorithm assigns the highest probability mass to the states with the lowest value of the value function. For this, the algorithm first sorts all states according to their current value of $V^{(n)}$ in ascending order and initializes $\mathbf{P}_{\text{inf}}$ with the lower bound, i.e., $\inf \mathbf{P}(s, \alpha, s'_i)$, of each possible transition. Next, it computes a *budget* representing the remaining probability mass to be distributed after subtracting all lower bounds, i.e., $\inf \mathbf{P}(s, \alpha, s')$, of each state $s' \in S'$. Now, as long as the budget is larger than the difference between the upper and lower bound of a state s'_i, it gets assigned its upper bound, i.e., $\sup \mathbf{P}(s, \alpha, s'_i)$. Once the budget is not sufficient anymore, the remaining budget moves towards the current state.

5.2.2 L_1-MDPs

The algorithm for L_1-MDPs follows a similar pattern as for IMDPs, i.e., assigning the states with the lowest value of $\underline{V}^{(n)}$ the highest probability mass. A dual version of this algorithm under the optimistic interpretation was introduced by Strehl and Littman in [38]. This algorithm was later adapted by Suilen et al. for the worst-case interpretation and is explicitly given in [39]. Since it is similar to Algorithm 1, we only provide an intuitive explanation here.

The algorithm begins by sorting all successor states in ascending order of $V^{(n)}$, yielding S'. The first state $s'_1 \in S'$ receives its probability mass from the center transition function $\tilde{\mathbf{P}}(s, \alpha, s'_1)$, plus half of the L_1-error, i.e., $\frac{d(s,\alpha)}{2}$. Next, all remaining states are assigned their probabilities according to $\tilde{\mathbf{P}}$. To ensure that $\mathbf{P}_{\text{inf}}$ remains a valid distribution, the algorithm subtracts up to $\frac{d(s,\alpha)}{2}$, starting at the highest valued states.

6 Conclusion

In this paper, we introduced Markov decision processes (MDPs), a widely used probabilistic model in both formal methods [5, 20, 16] and AI [42]. We discussed the reach-reward objective and showed how it can be solved using dynamic programming techniques, particularly through value iteration.

Building on this foundation, we explored robust Markov decision processes (RMDPs), which extend MDPs by incorporating uncertainty in the transition function. We explained the semantics of RMDPs as a game between an agent and nature, and classified them based on the structure of their uncertainty sets. We presented two important instances of RMDPs, interval MDPs (IMDPs) and L_1-MDPs, which are both (s, α)-rectangular and commonly used in formal verification and AI [39]. For these models, we showed how to adapt standard dynamic programming methods to account for worst-case resolution of uncertainty. We also presented concrete algorithms to solve the resulting inner minimization problems. These results complete the robust dynamic programming framework for solving reach-reward objectives in RMDPs using robust value iteration. To illustrate the key concepts, we used a running example involving a drone collecting deliveries until reaching its target, which we implemented as an MDP and IMDP using the Prism language and solved them by utilizing Storm and Prism.

Future work includes improving tool support for RMDPs, which is still limited. While IMDP functionality is partially available, L_1-MDPs are not yet supported in prominent tools such as Storm or Prism. Enhancing this support would enable the evaluation of theoretical advancements by executing benchmarks, which are still absent [39]. Furthermore, multi-environment MDPs (MEMDPs), which we did not cover in detail, are another interesting instance of RMDPs. They share similarities with partially observable MDPs (POMDPs) [39], a model class extensively studied in formal methods and AI [13, 45, 40, 11, 37, 41].

References

[1] The PRISM language, 2025. `https://www.prismmodelchecker.org/manual/ThePRISMLanguage/Introduction`.

[2] A. Abate, M. Prandini, J. Lygeros, and S. Sastry. Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Automatica*, 44(11):2724–2734, Nov. 2008.

[3] P. Ashok, J. Křetínský, and M. Weininger. PAC Statistical Model Checking for Markov Decision Processes and Stochastic Games. In *Computer Aided Verification*, pages 497–519, Cham, 2019. Springer International Publishing.

[4] T. Badings, L. Romao, A. Abate, and N. Jansen. Probabilities Are Not Enough: Formal Controller Synthesis for Stochastic Dynamical Models with Epistemic Uncertainty. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(12):14701–14710, June 2023.

[5] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[6] C. Baier, J. Klein, L. Leuschner, D. Parker, and S. Wunderlich. Ensuring the Reliability of Your Model Checker: Interval Iteration for Markov Decision Processes. In *Computer Aided Verification*, volume 10426, pages 160–180. Springer International Publishing, Cham, 2017.

[7] N. Balaji, S. Kiefer, P. Novotný, G. A. Pérez, and M. Shirmohammadi. On the Complexity of Value Iteration (Track B: Automata, Logic, Semantics, and Theory of Programming). *LIPIcs, Volume 132, ICALP 2019*, 132:102:1–102:15, 2019.

[8] E. Bartocci, J. Desharnais, P. Lindner, and A. Sokolova. A Probabilistic Analysis of Simplified Cluedo with Storm: The Birthday Cake Case. In *Principles of Verification: Cycling the Probabilistic Landscape*, volume 15261, pages 75–97. Springer Nature Switzerland, Cham, 2025.

[9] R. Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

[10] E. M. Bovy, M. Suilen, S. Junges, and N. Jansen. Imprecise Probabilities Meet Partial Observability: Game Semantics for Robust POMDPs. In *Proceedings of the Thirty-ThirdInternational Joint Conference on Artificial Intelligence*, pages 6697–6706, Jeju, South Korea, Aug. 2024. International Joint Conferences on Artificial Intelligence Organization.

[11] E. Boyarski, A. Felner, D. Harabor, P. J. Stuckey, L. Cohen, J. Li, and S. Koenig. Iterative-deepening conflict-based search. In C. Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4084–4090. International Joint Conferences on Artificial Intelligence Organization, 7 2020.

[12] S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge New York Melbourne New Delhi Singapore, version 29 edition, 2023.

[13] D. Braziunas and C. Boutilier. Stochastic local search for pomdp controllers. In *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, page 690–696. AAAI Press, 2004.

[14] F. Delgrange, J.-P. Katoen, T. Quatmann, and M. Randour. Simple Strategies in Multi-Objective MDPs. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 346–364, Cham, 2020. Springer International Publishing.

[15] N. Fijalkow, N. Bertrand, P. Bouyer-Decitre, R. Brenguier, A. Carayol, J. Fearnley, H. Gimbert, F. Horn, R. Ibsen-Jensen, N. Markey, B. Monmege, P. Novotný, M. Randour, O. Sankur, S. Schmitz, O. Serre, and M. Skomra. *Games on Graphs*. arXiv, 2023.

[16] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated Verification Techniques for Probabilistic Systems. In *Formal Methods for Eternal Networked Software Systems*, volume 6659, pages 53–113. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[17] E. M. Hahn, V. Hashemi, H. Hermanns, M. Lahijanian, and A. Turrini. Interval Markov Decision Processes with Multiple Objectives: From Robust Strategies to Pareto Curves. *ACM Transactions on Modeling and Computer Simulation*, 29(4):1–31, Oct. 2019.

[18] E. M. Hahn, V. Hashemi, H. Hermanns, and A. Turrini. Exploiting Robust Optimization for Interval Probabilistic Bisimulation. In *Quantitative Evaluation of Systems*, volume 9826, pages 55–71. Springer International Publishing, Cham, 2016.

[19] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, Sept. 1994.

[20] A. Hartmanns, S. Junges, T. Quatmann, and M. Weininger. A Practitioner's Guide to MDP Model Checking Algorithms. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 469–488, Cham, 2023. Springer Nature Switzerland.

[21] A. Hartmanns and B. L. Kaminski. Optimistic Value Iteration. In *Lecture Notes in Computer Science*, pages 488–511. Springer International Publishing, Cham, 2020.

[22] V. Hashemi, H. Hatefi, and J. Krčál. Probabilistic Bisimulations for PCTL Model Checking of Interval MDPs. *Electronic Proceedings in Theoretical Computer Science*, 145:19–33, Mar. 2014.

[23] V. Hashemi, H. Hermanns, L. Song, K. Subramani, A. Turrini, and P. Wojciechowski. Compositional Bisimulation Minimization for Interval Markov Decision Processes. In *Language and Automata Theory and Applications*, volume 9618, pages 114–126. Springer International Publishing, Cham, 2016.

[24] V. Hashemi, A. Turrini, E. M. Hahn, H. Hermanns, and K. Elbassioni. Polynomial-Time Alternating Probabilistic Bisimulation for Interval MDPs. In *Dependable Software Engineering. Theories, Tools, and Applications*, pages 25–41, Cham, 2017. Springer International Publishing.

[25] L. Heck, P. Leerkes, I. Melse, S. Junges, and M. Volk. Stormvogel documentation, 2025. `https://moves-rwth.github.io/stormvogel/1-introduction/1-Introduction.html`.

[26] C. Hensel, S. Junges, J.-P. Katoen, T. Quatmann, and M. Volk. The probabilistic model checker Storm. *International Journal on Software Tools for Technology Transfer*, 24(4):589–610, Aug. 2022.

[27] C. P. Ho, M. Petrik, and W. Wiesemann. Partial Policy Iteration for L1-Robust Markov Decision Processes. *J. Mach. Learn. Res.*, 22:275:1–275:46, 2021.

[28] G. N. Iyengar. Robust Dynamic Programming. *Mathematics of Operations Research*, 30(2):257–280, May 2005.

[29] B. Jonsson and K. Larsen. Specification and refinement of probabilistic processes. In *[1991] Proceedings Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 266–277, Amsterdam, Netherlands, 1991. IEEE Comput. Sco. Press.

[30] A. Klenke. *Probability Theory: A Comprehensive Course*. Universitext. Springer International Publishing, Cham, 2020.

[31] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Computer Aided Verification*, pages 585–591, Berlin, Heidelberg, 2011. Springer.

[32] P. Lindner. Github repository of the drone example, 2025. `https://github.com/plindnercs/trends-in-model-checking-drone-example`.

[33] A. Nilim and L. El Ghaoui. Robust Control of Markov Decision Processes with Uncertain Transition Matrices. *Operations Research*, 53(5):780–798, Oct. 2005.

[34] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57, 1977.

[35] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Number v.414 in Wiley Series in Probability and Statistics. John Wiley & Sons, Inc, Hoboken, 1994.

[36] T. Quatmann and J.-P. Katoen. Sound Value Iteration. In *Computer Aided Verification*, volume 10981, pages 643–661. Springer International Publishing, Cham, 2018.

[37] T. D. Simão, M. Suilen, and N. Jansen. Safe policy improvement for pomdps via finite-state controllers. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023.

[38] A. L. Strehl and M. L. Littman. An analysis of model-based Interval Estimation for Markov Decision Processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, Dec. 2008.

[39] M. Suilen, T. Badings, E. M. Bovy, D. Parker, and N. Jansen. *Robust Markov Decision Processes: A Place Where AI and Formal Methods Meet*, pages 126–154. Springer Nature Switzerland, Cham, 2025.

[40] M. Suilen, N. Jansen, M. Cubuktepe, and U. Topcu. Robust policy synthesis for uncertain pomdps via convex optimization. In C. Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4113–4120, 7 2020. Main track.

[41] M. Suilen, M. van der Vegt, and S. Junges. A PSPACE algorithm for almost-sure rabin objectives in multi-environment mdps. In R. Majumdar and A. Silva, editors, *35th International Conference on Concurrency Theory, CONCUR 2024, September 9-13, 2024, Calgary, Canada*, volume 311 of *LIPIcs*, pages 40:1–40:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[42] R. S. Sutton and A. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts, nachdruck edition, 2014.

[43] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis. Gymnasium: A Standard Interface for Reinforcement Learning Environments. Nov. 2024.

[44] W. Wiesemann, D. Kuhn, and B. Rustem. Robust Markov Decision Processes. *Mathematics of Operations Research*, 38(1):153–183, Feb. 2013.

[45] K. Wray and S. Zilberstein. A POMDP Formulation of Proactive Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Mar. 2016.